Dessert Pixel

Proyecto Fin de Grado - DAM

Gonzalo Solís Campos y Álvaro Bernárdez Hernández

unir LA UNIVERSIDAD EN INTERNET

ÍNDICE

Agradecimientos

Queremos expresar nuestro más sincero agradecimiento por estos años de estudio en el grado superior de Desarrollo de Aplicaciones Multiplataforma. Ha sido una experiencia gratificante y enriquecedora que ha dejado una huella imborrable en nuestra vida.

Queremos agradecer a la institución educativa por proporcionarnos un buen entorno para aprender y crecer, así como por ofrecernos todos los recursos y herramientas que han enriquecido nuestra experiencia académica.

No podemos olvidar mencionar a nuestras familias y seres queridos, quienes han estado a nuestro lado durante todo este tiempo. Gracias por su comprensión y por motivarnos a seguir adelante.

Por último, pero no menos importante, una pequeña mención para nosotros por nuestra dedicación, perseverancia y compromiso a lo largo de estos años que nos han mantenido unidos con el objetivo de finalizar este grado. Han sido muchas horas de estudio, trabajo y sacrificio, pero cada esfuerzo ha valido la pena.

En definitiva, **GRACIAS** por hacer de estos años una experiencia inolvidable.

1. Memoria del proyecto

El presente documento constituye la memoria del proyecto de fin de grado en Desarrollo de Aplicaciones Multiplataforma.

El objetivo principal de este proyecto ha sido crear un videojuego de plataformas en 2D, innovador y entretenido, que aproveche al máximo las capacidades que ofrecen las diferentes plataformas existentes, en especial los dispositivos móviles y los ordenadores.

No está orientado a un público con una edad concreta, pero se busca que éste experimente nostalgia al jugar este tipo de género. A lo largo de esta memoria, se describe en detalle el proceso seguido para lograr este propósito, abordando desde la concepción inicial del juego hasta su implementación final.

La elección del tema de este proyecto ha sido poder mejorar uno de los trabajos que se propuso en clase, en el cual vimos un gran potencial y que además se podía implementar para múltiples plataformas como Windows, IOS, Android o Web.

En resumen, esta memoria representa el registro completo del proyecto de fin de grado en Desarrollo de Aplicaciones Multiplataforma, centrado en el diseño y el desarrollo de un videojuego de plataformas interactivo en 2D.

A través de las siguientes secciones, se brindará una visión integral del proceso seguido, los retos enfrentados y los logros obtenidos. Esperamos que esta memoria refleje el esfuerzo y la dedicación invertidos en la creación de un videojuego que brinde una experiencia divertida, envolvente y desafiante para sus usuarios.

2. Módulos formativos aplicados en el TFG

Durante el desarrollo de este TFG, se han aplicado los módulos formativos que se detallan a continuación.

Programación:

Esta asignatura es la más influyente en el proyecto junto con programación multimedia. Se explica más adelante cómo se ha distribuido el trabajo y con qué lenguajes.

Programación multimedia y dispositivos móviles:

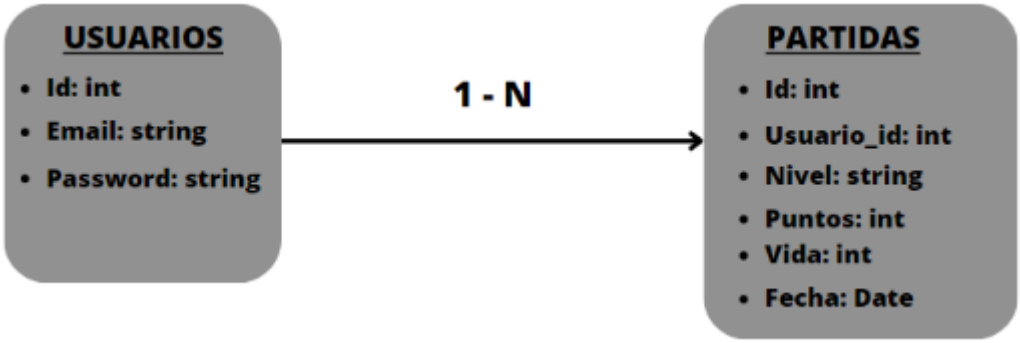
Otra asignatura clave para el desarrollo del videojuego. Donde aprendimos el uso de la herramienta **Unity** y otros lenguajes de programación como **C#**, que constituyen la base principal de este proyecto. También aprendimos cómo utilizar servicios que aprovechen las capacidades multimedia de diferentes dispositivos.

Entornos de desarrollo:

Esta asignatura brindó los conocimientos de **Git** básicos para estructurar y desarrollar un proyecto conjunto. Se tuvo que aprender algunos comandos extras para perfeccionar el uso de la herramienta **Git** además del uso de **Github Desktop** que ha felicitado mucho la tarea de control de versiones.

Bases de Datos:

Esta asignatura brindó los conocimientos sobre persistencia y modelado de datos necesarios para estructurar y desarrollar nuestro proyecto. Para este proyecto, hemos utilizado un modelo entidad relación 1-N.



Acceso a datos:

Esta asignatura nos proporcionó los conocimientos necesarios sobre la integración de bases de datos en proyectos multiplataforma. Para este proyecto, hemos utilizado un plugin de **SQlite** diseñado para Unity, que permite la integración de este tipo de base de datos con videojuegos desarrollados en Unity para Android, Windows e IOS-

Lenguaje de marcas:

Esta asignatura está relacionada con el desarrollo web y se centra en el estudio y aplicación de lenguajes de marcado utilizados para estructurar y presentar información en la web.

Desarrollo de interfaces:

Se relaciona con crear interfaces de usuario efectivas, atractivas y funcionales para aplicaciones y sistemas interactivos, teniendo en cuenta aspectos técnicos, estéticos y de usabilidad.

3. Herramientas/Lenguajes Utilizados

Unity:

Es una plataforma de desarrollo de juegos y aplicaciones en 2D y 3D que ofrece un conjunto completo de herramientas para crear experiencias interactivas en diferentes plataformas como Microsoft Windows, Mac OS, Linux, Android e Ios. Es una herramienta ampliamente utilizada en la industria del desarrollo de videojuegos.

Integra un editor de versiones a modo de interfaz gráfica de usuario (GUI) que permite a los desarrolladores crear, editar y organizar todos los elementos que integran un proyecto de Unity, como escenas, objetos, componentes, scripts y assets. Este proyecto se ha realizado con la versión **2020.3.18f1**

Github / Github Desktop:

Es una plataforma en línea que facilita la colaboración y el control de versiones de proyectos de desarrollo de software. Es un servicio de alojamiento de repositorios de código fuente basado en Git, un sistema de control de versiones distribuido.

C#:

Es un lenguaje de programación moderno y de propósito general desarrollado por Microsoft. Está diseñado para ser un lenguaje orientado a objetos y se basa en el marco de trabajo .NET. Esto permite que los programas escritos en **C#** se ejecuten en diferentes sistemas operativos, como Windows, macOS y Linux

SQLite:

Es un sistema de gestión de bases de datos relacional que se distingue por su ligereza y autonomía. Se trata de una biblioteca de código abierto que se incorpora directamente en las aplicaciones, sin necesidad de un servidor independiente. Esto significa que todas las funciones de gestión de bases de datos están integradas en la propia aplicación que lo utiliza. Su tamaño reducido y bajo consumo de recursos hacen que sea especialmente adecuado para aplicaciones con limitaciones de espacio o rendimiento.

La principal utilidad de SQlite reside en su capacidad para almacenar y administrar datos de forma eficiente en aplicaciones que requieren una base de datos local. Su portabilidad permite que se ejecute en diversas plataformas, incluyendo sistemas operativos de escritorio y dispositivos móviles. Además, SQlite ofrece soporte para transacciones ACID, garantizando la integridad y consistencia de los datos. También proporciona una amplia gama de operaciones SQL estándar, permitiendo la creación, actualización y eliminación de tablas, así como consultas avanzadas de selección y filtrado de datos. En resumen, SQlite es una solución de gestión de bases de datos relacional compacta y autónoma.

Visual Studio:

Proporciona un conjunto completo de herramientas y características para facilitar el desarrollo de software en diferentes lenguajes de programación, incluyendo **C#**, **C++**, **VB.NET**, **Python** y más.

También ofrece una interfaz gráfica de usuario (GUI) y un conjunto de herramientas que permiten a los desarrolladores escribir, depurar, compilar, probar y desplegar aplicaciones de software de manera eficiente.

HTML:

Es el lenguaje fundamental que se utiliza para crear páginas web, ya que es interpretado por todos los navegadores web para mostrar su contenido, visualmente.

CSS:

Permite controlar la apariencia de los elementos HTML, como el color, la fuente, el tamaño, la disposición y otros atributos visuales.

JavaScript:

Permite agregar interactividad y dinamismo a las páginas web, mediante la manipulación del contenido y la respuesta a eventos. Se ejecuta en el navegador del cliente y se utiliza para realizar acciones como validar formularios, modificar el contenido de una página en tiempo real, interactuar con servicios web y crear efectos visuales.

Bootstrap:

Proporciona una amplia colección de componentes prediseñados, como botones, barras de navegación, tarjetas y formularios, que se pueden utilizar fácilmente en el desarrollo de páginas web

4. Fases del proyecto

4.1 Idea

Se comienza con el desarrollo de la idea principal, mejorar uno de los trabajos que se propuso en clase, el cual nos pareció que podía tener un gran potencial como trabajo de fin de grado.

Como primera entrega se desarrolló un anteproyecto en el que se exponía en qué consistiría el desarrollo de nuestro videojuego y hasta donde pretendíamos llegar:

“Nuestra idea consiste en desarrollar un videojuego de plataformas en 2D, con al menos 5 niveles diferentes, en el que los jugadores podrán poner a prueba sus habilidades, cumpliendo con distintos objetivos como la recogida de distintos coleccionables para aumentar de nivel durante el desarrollo del juego y su enfrentamiento con distintos tipos de enemigos para poder superar cada nivel.

El juego contará con tres versiones: Una versión para escritorio, otra versión para móvil y una versión para web. Además, el juego contará con una página web oficial, desde la que se podrá descargar el juego en todas sus versiones e incluso jugar directamente.”

Se ha requerido una planificación cuidadosa, dedicación y habilidades en diferentes áreas. Además, ha sido fundamental establecer objetivos realistas. Desde la programación hasta el diseño de los niveles, cada aspecto del desarrollo del videojuego se ha adaptado para cumplir los objetivos establecidos.

Se puede dividir el proyecto en 3 fases:

1ª Parte:

4.2 Mecánicas

Dada la idea principal, lo siguiente fue decidir cuales iban a ser las mecánicas que iba a tener el protagonista, para poder continuar con el desarrollo del juego. Se decidió empezar por el desarrollo del Jugador; y se decidió también que la mecánica principal fuesese simple, **saltar**. Con eso en mente, lo primero que te viene a la cabeza es el clásico videojuego de “Super Mario”, pero para asegurarnos de cumplir con los objetivos propuestos, se decidió eliminar los “power ups” del alcance del proyecto.

No obstante, se echaba en falta una mecánica que diera más dinamismo a la jugabilidad, por ello se nos vino a la cabeza otro clásico, el “Crash Bandicoot”, aun así, se quería conservar que la mecánica principal fuera el salto, asique se decidió finalmente eliminar la idea de una animación de ataque, como por ejemplo el clásico torbellino, propio de dicho juego, pero se mantuvo la idea del doble salto y el sistema de vidas del “Crash Bandicoot 4: It's About Time”.

A continuación, se adjunta un fragmento de código con las mecánicas del salto y el doble salto.

```
public override void Tick()
{
    //Comprobamos si el jugador esta en el suelo
    base.Tick();
    jumpingTime += Time.deltaTime;
    //Si el jugador pulsa la tecla W, saltamos
    if (Input.GetKeyDown(KeyCode.W)) DoubleJump();
    // Si el tiempo de salto es inferior o igual a 0.2f, el código no se ejecutará
    //Esto sirve para que el jugador no pueda saltar infinitamente
    if (jumpingTime <= 0.2f) return;
    //Raycast para comprobar si el jugador esta en el suelo
    RaycastHit2D hit = Physics2D.Raycast((base.controller.PlayerFeet.position), -base.controller.Tr.up, 10, groundLayer);
    //Debug
    Debug.DrawRay((base.controller.PlayerFeet.position),-base.controller.Tr.up,Color.blue,1);
    //Si el raycast ha colisionado con el suelo, cambiamos el estado a PlayerGrounded
    if (hit)
    {
        if (hit.distance <= 1) //0 es un valor demasiado bajo para detectar la colision
        {
            base.controller.ChangeState(PlayerStatesEnum.PlayerGrounded);
            base.controller.AnimationController.SetTrigger(groundedHash);
        }
    }
}
```

```
/* Método DoubleJump */
private void DoubleJump()
{
    //Si el numero de saltos es mayor que 2, el código no se ejecutará
    if (numberOfJumps >= 2) return;
    // Se ejecuta el sonido
    base.controller.GetComponent<PlayerController>().PlaySound("jump");
    //Velocidad del salto
    base.controller.Rb.velocity = Vector2.zero;
    //Animacion de salto
    base.controller.AnimationController.SetTrigger(jumpHash);
    //Aumentamos el numero de saltos
    numberOfJumps++;
    //Se ejecuta el salto
    base.controller.Rb.AddForce(Vector2.up * (jumpForce), ForceMode2D.Impulse);
}
```

El siguiente paso fue pensar en los enemigos, cada uno con sus respectivas animaciones. Se decidió desarrollar múltiples enemigos para que el escalado de dificultad fuese equilibrado y notable. Así mismo, se recurrió a los patrones típicos de éstos, en los juegos mencionados anteriormente.

Cada tipo de enemigo usa su propio script: Assets \ Scripts \ Enemy.

- Enemigos que persiguen al jugador.

```
/* Método PerseguirPlayer*/
public void PerseguirPlayer()
{
    // Hacemos que el enemego vaya desde su posición hasta la del player y aumentamos su velocidad
    transform.position = Vector3.MoveTowards(transform.position, playerController.transform.position, speed * Time.deltaTime);
    // Ajustamos la dirección para que mire al jugador
    Vector3 directionToPlayer = playerController.transform.position - transform.position;
    if (directionToPlayer.x < 0)
    {
        tr.localScale = new Vector3(1, 1, 1);
    }
    else
    {
        tr.localScale = new Vector3(-1, 1, 1);
    }
}
```

- Enemigos que patrullan cambiando de dirección cuando colisionan con un obstáculo.

```
/* Método FixedUpdate */
void FixedUpdate()
{
    hit = Physics2D.Raycast((tr.position), direction, 2, groundLayer);
    Debug.DrawRay((tr.position), direction, Color.blue, 1);
    if (hit)
    {
        if (hit.distance <= 0.5f)
        {
            ChangeDirection();
        }
    }
}
```

```
/* Método ChangeDirection */
protected void ChangeDirection()
{
    if (direction == Vector3.right)
    {
        direction = -Vector3.right;
        tr.localScale = new Vector3(1, 1, 1);
    }
    else
    {
        direction = Vector3.right;
        tr.localScale = new Vector3(-1, 1, 1);
    }
}
```

- Enemigos que disparan proyectiles al jugador.

```
/* Método Shoot */
private void Shoot()
{

    // Si el jugador está dentro del rango de disparo y ha pasado el tiempo de espera para el siguiente disparo
    if (IsPlayerNearEnemy() && (Time.time >= nextShootTime))
    {
        // Crear una instancia del prefab de bala en la posición del firePoint
        bullet = Instantiate(bulletPrefab, firePoint.position, Quaternion.identity);
        GameManager.Instance.PlaySound("boom");
        SetBulletDirecction();
        // Establecer el tiempo de espera para el siguiente disparo
        nextShootTime = Time.time + shootInterval;
    }
```

- Enemigos que patrullan y persiguen al jugador cuando detectan que este está cerca.

```
/* Método IsPlayerNearEnemy */
private bool IsPlayerNearTrap()
{
    if (playerController.Life <= 0 || playerController.IsPlayerWinner == true) return false;
    float distance = (Math.Abs(transform.position.x) - Math.Abs(playerController.transform.position.x));
    if (Math.Abs(distance) <= maxDistance)
    {
        return true;
    }
    else
    {
        return false;
    }
```

Por último, se determinó usar plataformas móviles para agregar variedad y desafío a la experiencia de juego. Además, este es un recurso ya utilizado en clase y que nos pareció buena idea reutilizar. Todos sus scripts se encuentran en la ruta Assets \ Scripts.

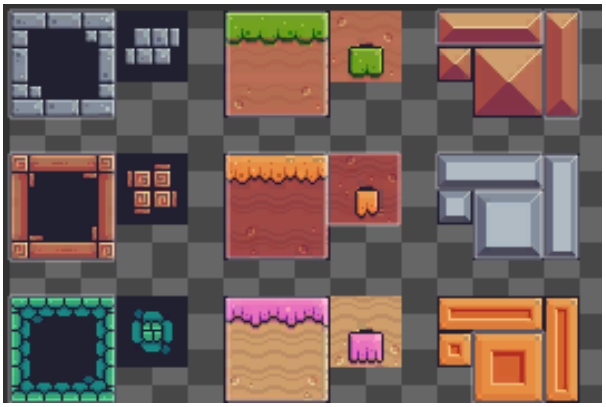
```
/* Método MoveHorizontal */
void MoveHorizontal()
{
    // Cambiar la dirección de movimiento si el objeto alcanza el límite de distancia
    if ((transform.position.x >= (startingPosition + distance/2f))
    || (transform.position.x <= (startingPosition - distance/2f))) {
        direction = -direction;
    }

    // Calcular la nueva posición del objeto
    transform.position += new Vector3((direction * speed * Time.deltaTime), 0f, 0f);
}
/* Método MoveVertical */
void MoveVertical()
{
    // Cambiar la dirección de movimiento si el objeto alcanza el límite de distancia
    if ((transform.position.y >= (startingPosition + distance/2f))
    || (transform.position.y <= (startingPosition - distance/2f))) {
        direction = -direction;
    }

    // Calcular la nueva posición del objeto
    transform.position += new Vector3(0f, (direction * speed * Time.deltaTime), 0f);
```

4.3 Diseño de niveles

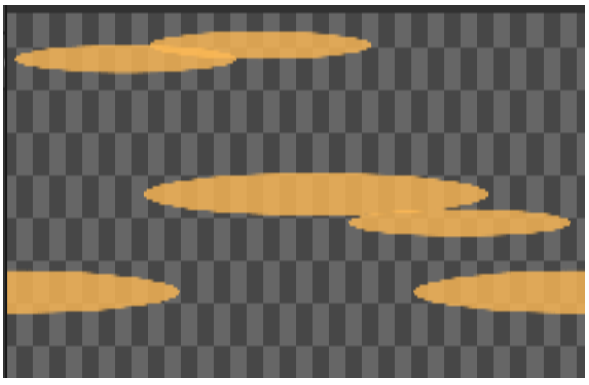
Con las mecánicas ya claras y definidas el siguiente paso fue la creación de niveles. El diseño de éstos influye directamente en la experiencia del jugador. Un nivel bien diseñado proporciona desafíos interesantes y equilibrados y brinda satisfacción cuando se superan los obstáculos.

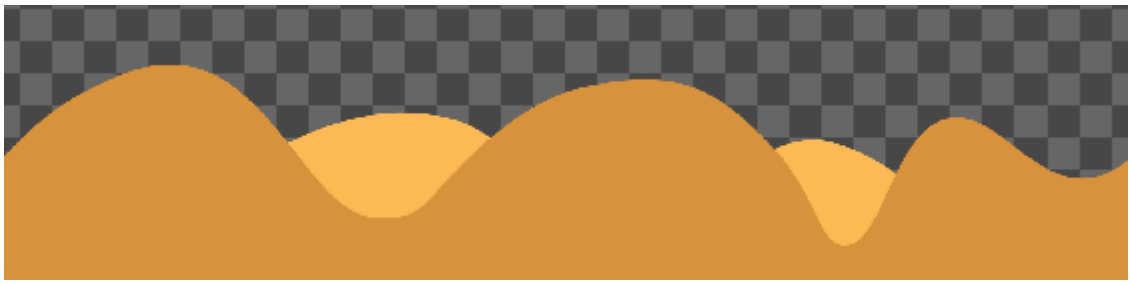
El principal objetivo ha sido plantear niveles que permitan una progresión suave a lo largo del juego. Se presentan gradualmente nuevos desafíos y mecánicas, lo que garantiza una experiencia de juego fluida y coherente. Se ha intentado crear variedad para evitar que el juego se vuelva monótono y al mismo tiempo ajustarnos al periodo establecido para terminar el proyecto.

Dentro de la carpeta de sprites se ha utilizado la siguiente paleta para la creación de dichos escenarios:

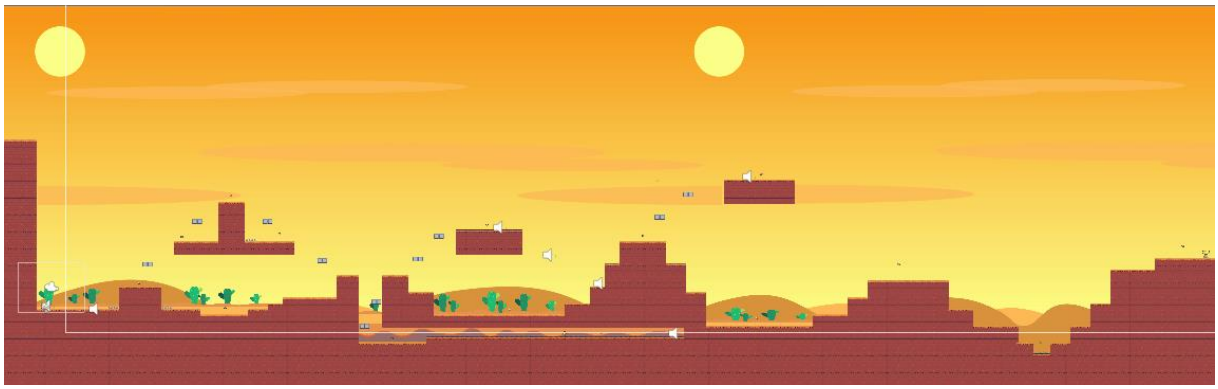


Y estos son algunos de los backgrounds usados:



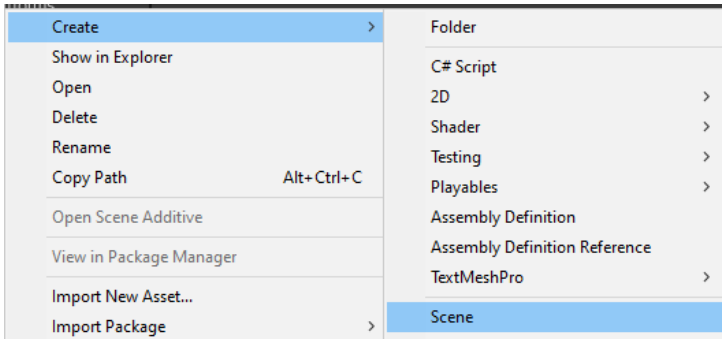


A continuación, se muestra el primer nivel usando todo lo anterior:



Todos los niveles fueron creados como nuevas escenas y se encuentran en la ruta:

Assets \ Scenes.



4.4 Puntos y coleccionables:

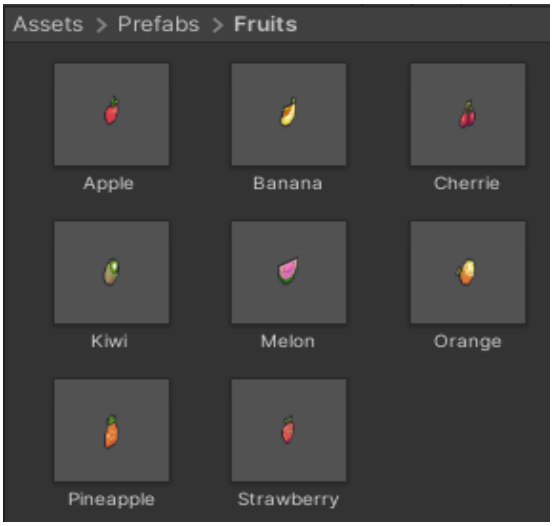
Una vez desarrollados los niveles se pensó en el sistema de puntos y coleccionables. Dentro del juego se pueden obtener puntos matando enemigos y recogiendo las frutas y coleccionables que se encuentran a lo largo de los niveles. Cada vez que derrotes a un enemigo, serás recompensado con puntos. A medida que se avance en el juego los diferentes enemigos tendrán más vida, pero también más puntos.

MUSHROOM	**SLIME**	**RABBIT**
Vidas: 1	Vidas: 1	Vidas: 3
Puntos: 10	Puntos: 20	Puntos: 30
Velocidad: 0.6	Velocidad: 0.46	Velocidad: 1.05

Cada nivel estará lleno de frutas, unas serán más difíciles de conseguir que otras ya que te dan más puntos al recolectarlas. Todos los prefabs relativos a estos coleccionables se encuentran en la siguiente ruta: Assets \ Prefabs \ Fruits.

Dichas frutas son:

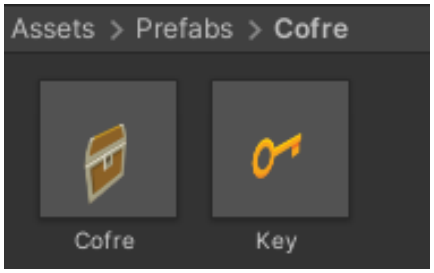
- Cereza: 5 puntos
- Kiwi: 10 puntos
- Manzana: 15 puntos
- Plátano: 20 puntos
- Naranja: 25 puntos
- Melón: 30 puntos
- Piña: 35 puntos



También cuentan con su propia animación y un script para que desaparezcan cuando el jugador las recolecta. Assets \ Scripts \ Coleccionables.

```
/* Método OnTriggerEnter2D */
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.gameObject.CompareTag("Player"))
    {
        // Ejecutamos la animación collected
        anim.SetTrigger("collected");
        // Emitimos el sonido de recolección
        player.PlaySound("fruit");
        // Añadimos los puntos al jugador
        player.AddPoints(points);
        // Destruimos la fruta
        Invoke("DestroyFruit", timeBeforeDestroy);
    }
}
```

En cada nivel se encuentra un coleccionable en forma de cofre. Para poder abrir dichos cofres y conseguir los puntos que estos proporcionan, se necesita adicionalmente una llave que se encuentra en otra parte del mapa. Al abrirlos el jugador obtendrá 100 puntos. Se encuentran en la carpeta de: Assets \ Prefabs \ Cofre.



Tanto la llave como el cofre cuentan con sus propias animaciones y un script para poder coger la llave y abrir el cofre. Assets \ Scripts \ Cofres.

```
private void OnTriggerEnter2D(Collider2D other)
{
    if(other.gameObject.CompareTag("Player"))
    {
        if((other.GetComponent<PlayerController>().PlayerHasKey) && (!isOpened))
        {
            audioSource.Play();
            other.GetComponent<PlayerController>().AddPoints(points);
            anim.SetTrigger("Open");
            isOpened = true;
        }
        SpriteRenderer.sortingOrder = backOrderInlayer;
    }
}
```

4.5 HUD:

Para terminar esta primera parte del desarrollo, implementamos un HUD, que consiste en información gráfica, repartida por la pantalla que se visualiza durante el juego, por ejemplo, en este caso se muestran los puntos y la información relativa al nivel de vida del jugador. Dicha información se coloca bien repartida en los extremos de la pantalla para que no interfiera perjudicialmente en la experiencia del usuario.

Los puntos se van actualizando a medida que se recogen frutas, se matan enemigos o se abren los cofres. El nivel de vida varía dependiendo de si el jugador es golpeado por un enemigo o una trampa o si este recolecta más vidas, pudiendo bajar o subir. Los jugadores empezarán la partida con 3 vidas y 0 puntos, si las vidas del jugador llegan a 0, aparecerá la típica pantalla de GAME OVER y el este deberá volver a comenzar el nivel.

A continuación, se adjunta un ejemplo de cómo incrementan los puntos cuando el jugador recoge las frutas:

```
/* Método OnTriggerEnter2D */
private void OnTriggerEnter2D(Collider2D other)
{
    if (other.gameObject.CompareTag("Player"))
    {
        // Ejecutamos la animación collected
        anim.SetTrigger("collected");
        // Emitimos el sonido de recolección
        player.PlaySound("fruit");
        // Añadimos los puntos al jugador
        player.AddPoints(points);
        // Destruimos la fruta
        Invoke("DestroyFruit", timeBeforeDestroy);
    }
}
```

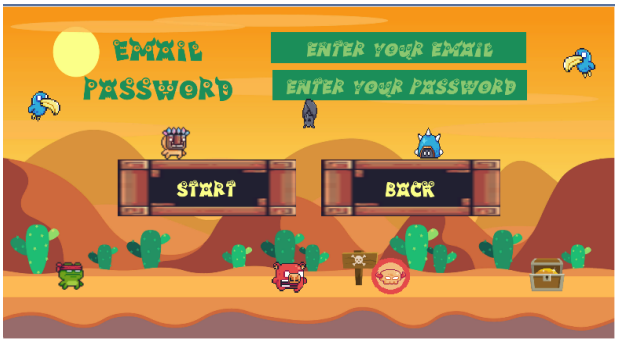
2ª Parte:

4.6 Menús:

Una vez llegados a este punto, se realizan todas las pruebas necesarias para que los niveles queden equilibrados, y los enemigos, los puntos, las vidas y las plataformas queden correctamente distribuidos. Después de comprobar que todos los componentes de la primera parte funcionaban correctamente, se desarrolló el menú principal, el cual se encuentran en la ruta: Assets \ Scenes \ Menú.

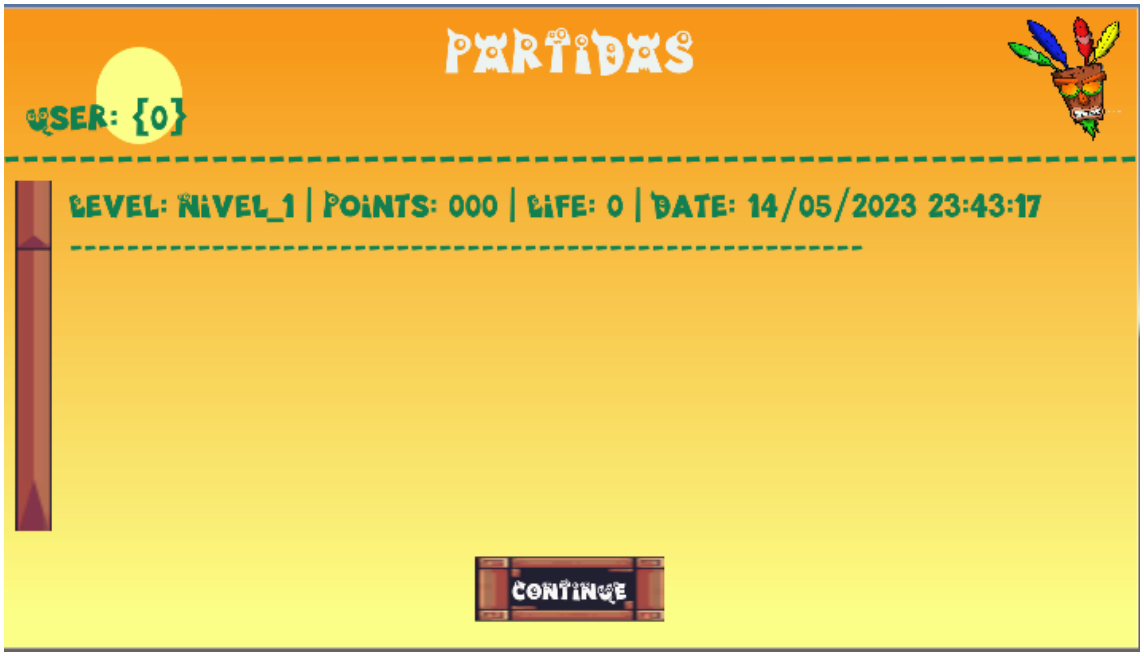
En dicho menú, el jugador tiene dos opciones: “New Game”, opción con la que se podrá empezar una nueva partida o “Load Game”, opción con la que se podrá continuar una partida ya empezada, en la que se guardan tanto los puntos como las vidas.
Dicha escena cuenta también con un botón en la parte superior derecha para salir directamente del juego.

En ambas opciones se deberá introducir un correo electrónico y una contraseña, tanto si se quiere continuar con una partida ya empezada, como si el jugador desea empezar una nueva partida. Si hay algún dato incorrecto se muestran mensajes de error.



Por otro lado, también disponemos de un panel de Game Over, oculto dentro del propio HUD, que te da la opción de reintentar el nivel o de salir al menú principal; y que solo se muestra cuando el jugador pierde todas las vidas.

También se ha desarrollado una escena final para el juego, donde se muestran los créditos. Una serie de escenas con diferentes animaciones para presentar de forma amena a los desarrolladores, los contenidos del juego y como no, dar gracias a los jugadores por haberlo terminado. También se visualizará una pantalla con un resumen de las partidas finalizadas o en progreso y sus puntuaciones, correspondientes al usuario logueado. Dicha escena se encuentra en la ruta: Assets \ Scenes \ Credits.



4.7 Base de datos:

Para poder integrar un modelo de persistencia de datos en nuestro videojuego hecho con unity, hemos seguido los siguientes pasos, no obstante, toda la implementación puede encontrarse en la ruta **Assets \ Scripts \ Bbdd** de nuestro proyecto.

1. **Descargar el plugin de SQLite:** Lo primero que hicimos fue descargar el plugin de **SQLite** para Unity. Hay varias opciones disponibles, pero una de las más populares es **SQLite4Unity3d**, que es un wrapper ligero y fácil de usar para **SQLite**. La última versión de este plugin se encuentra en su repositorio de GitHub.

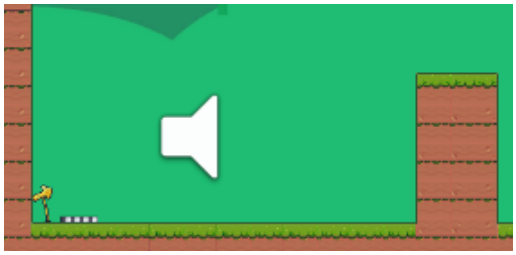
2. **Importar el plugin en Unity:** Para importar dicho plugin en nuestro proyecto de Unity, tuvimos que arrastrar los archivos descargados del repositorio GitHub de **SQLite4Unity3d** a la carpeta "Assets" de nuestro proyecto. Es importante asegurarse de que los archivos del plugin se encuentran en el directorio "Assets \ Plugins" del proyecto, para que Unity los reconozca correctamente.

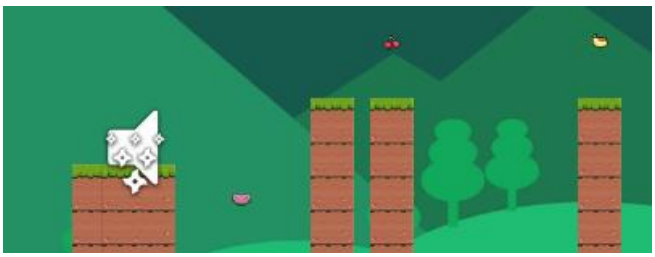
3. **Crear una base de datos SQLite:** Una vez que importamos el plugin, comenzamos a integrar las librerías de **SQlite** en nuestro proyecto. Para ello creamos un script específico para la integración de un CRUD completo llamado “DataService.cs” y creamos otros dos scripts más para representar las tablas que utilizamos en nuestro videojuego para persistir información, que son el script “Partida.cs” y el script “Usuario.cs”.

4. **Utilizar la base de datos en nuestro videojuego:** Una vez creados todos los métodos necesarios (CRUD completo: CREATE, READ, UPDATE, DELETE) en el script “DataService.cs”, usamos dichos métodos desde otros scripts de nuestro videojuego para guardar, cargar, borrar y actualizar datos relativos a los usuarios del videojuego y sus partidas (1-N).

4.8 Detalles, música y partículas:

Para terminar esta segunda parte del desarrollo, se añade la música de fondo para los distintos niveles, los sonidos para las animaciones (el salto o el doble salto), los sonidos para la recolección de las frutas y los sonidos de los enemigos.





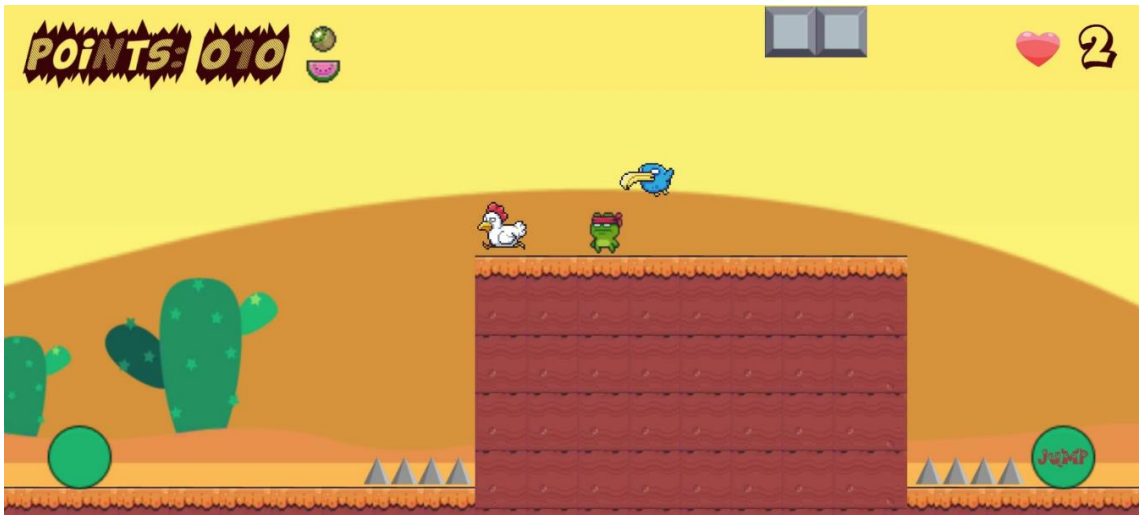
También se implementan nuevas animaciones para el jugador, cuando este recibe daño parpadea y se vuelve invulnerable durante un segundo, pero cuando se agotan todas sus vidas estalla un sistema de partículas.

3ª Parte:

4.9 Android - IOS:

Para adaptar nuestro videojuego a dispositivos móviles, utilizamos un recurso que proporciona Unity, llamado "Standard Assets" el cual incluye scripts y prefabs predefinidos para facilitar la implementación de controles para pantallas táctiles en un videojuego.

Una vez que importamos dicho recurso, solo tuvimos que adaptar los scripts del player, para que sus mecánicas se adaptaran a los controles táctiles que se muestran en pantalla.



4.10 WEB:

La última parte del proyecto ha sido desarrollar una web con el objetivo de proporcionar información general del juego, Jugabilidad, Niveles y Enemigos. En dicha web también pueden verse imágenes del juego, de los niveles y de los enemigos de forma secuencial para mostrar una idea más visual del contenido del juego.

En la parte inferior se puede probar una demo del juego, pero solo si el navegador web se ha abierto desde un dispositivo con una resolución de pantalla de mínimo 1020 px de ancho. Esta demo brinda comodidad y accesibilidad a los jugadores al poder probar una pequeña parte del juego antes de adquirir la versión completa. Además, incluye la opción de poder jugar a pantalla completa, para satisfacer a los usuarios más gamers.

Desde la misma página web se da la opción de descargar el juego completo tanto para Android como para Windows:

El objetivo es que se convierta en un punto central donde los jugadores puedan obtener información, probar el juego y acceder a su versión descargable, lo que mejora la experiencia del usuario y aumenta las posibilidades de éxito del juego.

5. Conclusiones y mejoras del proyecto

Se ha conseguido cumplir los objetivos planteados al comienzo del TFG. Se ha conseguido que el proyecto funcione en las diferentes plataformas de Windows, Android, IOS y Web.

El desarrollo del proyecto ha permitido adquirir un amplio conocimiento sobre el campo de las aplicaciones multiplataforma. Se debe destacar que, debido al tiempo dado para la realización del proyecto, se tomaron una serie decisiones para acortar el contenido del juego y obtener una lista de tareas real que permitiera entregarlo a tiempo sin sacrificar la idea principal del juego.

Las herramientas de control de versiones han sido un punto clave, ya que constantemente se modificaban Assets y partes del código que influían en el trabajo del equipo.

En conclusión, el desarrollo de este proyecto basado en un videojuego ha sido un éxito en términos de alcanzar los objetivos propuestos y adquirir conocimientos significativos en el campo del desarrollo de videojuegos y las aplicaciones multiplataforma. Se logró diseñar, desarrollar e implementar un videojuego funcional que ofrece una experiencia envolvente y desafiante para los jugadores. El objetivo principal de crear un juego entretenido, consideramos que se ha conseguido mediante el diseño cuidadoso de cada nivel y su dificultad.

Durante el proceso de desarrollo y las pruebas, se identificaron algunas áreas de mejora como, por ejemplo:

- El diseño de niveles podría mejorarse para ofrecer una curva de dificultad más equilibrada y un progreso más suave en el juego. Esto podría conseguirse añadiendo más niveles intermedios.
- El diseño de un jefe final para completar la historia.
- Explorar la incorporación de opciones multijugador para brindar una experiencia aún más enriquecedora a los jugadores.
- Implementar nuevas mecánicas y agregar logros.

6. Componentes del equipo

7. Bibliografía / Palabras clave

De forma genérica, se han utilizado las fuentes que se detallan a continuación:

- **Wikipedia** (https://es.wikipedia.org)
- **StackOverflow** (https://stackoverflow.com/)
- **GitHub** (https://github.com/)
- **Assets (**https://pixelfrog-assets.itch.io/pixel-adventure-1**) y (**https://pixelfrog-assets.itch.io/pixel-adventure-2**)**
- **SQlite** (https://github.com/robertohuertasm/SQLite4Unity3d)

Palabras clave:

- **Assets**: Son los recursos utilizados en el desarrollo de juegos y aplicaciones. Un asset es cualquier elemento que se puede importar y utilizar en tu proyecto, como modelos 3D, texturas, sonidos, scripts, animaciones, prefabs y más.
- **Prefab**: Es una plantilla o molde que contiene todos los componentes, configuraciones y propiedades de un objeto en particular.
- **Script**: Un archivo de código fuente que contiene instrucciones y algoritmos que definen el comportamiento y la lógica de un juego o una aplicación.
- **Sprite**: Es una imagen bidimensional que representa un objeto, personaje, elemento del entorno o cualquier otro elemento visual dentro de un videojuego. Son elementos gráficos básicos que se utilizan para construir las escenas y las animaciones en los videojuegos.
- **Power Up**: Es un elemento o acción que otorga a un personaje o jugador una mejora o ventaja temporal en el juego.
- **Sprites**: Son elementos gráficos 2D que se utilizan para representar personajes, objetos y elementos visuales en pantalla.
- **Backgrounds**: Se refiere a las capas o imágenes de fondo que se utilizan para representar el entorno o escenario en el que se desarrolla el juego.

8. Anexos

Tanto el código fuente de la aplicación, como los scripts utilizados y la base de datos pueden visualizarse en el repositorio de GitHub.

GitHub: (https://github.com/GonzaloSC95/TFG_DAM)

Web: (https://gonzalosc95.github.io/DESSERT_PIXEL/)